Juho Hildén

Äänen suoratoisto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotekniikka

Insinöörityö

7.5.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Juho Hildén Äänen suoratoisto 35 sivua 7.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Vesa Ollikainen, Lehtori Ville Kulmala, Ohjelmistosuunnittelija

Työssä tutustutaan suoratoistoon verkon yli sekä kehitetään verkon yli toimivan kuulutusjärjestelmän ohjainrajapinta ja sille käyttöliittymä. Käyttöliittymän tarkoitus on kuitenkin toimia testityökaluna ohjainrajapinnan toimintojen testaamiseen. Kumpikin on toteutettu Javalla käyttäen pohjana OBT-kuulutusjärjestelmän käyttäjärajapintaa.

Työn teoriaosuudessa esitellään suoratoiston käyttötarkoituksia ja siinä yleisesti käytettyjä tekniikoita. Tekniikoista tarkastellaan niiden käyttötarkoitusta, vahvuuksia ja heikkouksia sekä sitä, milloin mitäkin tekniikkaa on tarkoituksenmukaista käyttää ja esimerkiksi, milloin suoratoistossa on syytä käyttää UDP-protokollaa ja milloin TCP-protokollaa.

Teoriaosuuden jälkeen seuraa ohjainrajapinnan ja käyttöliittymän esittely, jossa esitellään niiden sisältämät toiminnot ja toimintojen toteutus koodiesimerkkien kanssa. Työn lopussa on yhteenveto ja arviointiosuus, jossa arvioidaan, kuinka ohjainrajapinnan ja käyttöliittymän toteutukset onnistuivat sekä mitä jatkokehitysmahdollisuuksia niissä voisi olla.

Työ on tehty Avack Oy:lle. Avack on tamperelainen viestintäratkaisutalo.

Avainsanat	Suoratoisto, Java

Author Title Number of Pages Date	Juho Hildén Sound Streaming 35 pages 7 May 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Vesa Ollikainen, Senior Lecturer Ville Kulmala, Software Developer

This thesis showcases streaming over internet as well as introduces a control interface for a public address system and a graphical user interface for that. The main purpose of the graphical user interface was to be a test environment for the control interface. Both were written with Java using the OBT public address system client interface as base. The study was done for Avack Oy. Avack is a company specialized in communication solutions.

The first section of this thesis is about the intended use and technologies of streaming. The technologies are viewed from the perspective of intended uses, strengths and weakness. Also there are examples when each technology should be used, such as when to use the TCP-protocol or the UDP-protocol in streaming.

The study also introduces the control interface and the graphical user interface, including descriptions of the different features of the program and how they are implemented with code examples. Next, the study describes the evaluation of how the implementations succeeded and how development could be continued.

Keywords	Streaming, Java

Sisällys

Lyhenteet

TCP	Transmission Control Protocol. Tiedonsiirtoprotokolla tavujonojen luotettavaan siirtämiseen kahden laitteen välillä.
UDP	User Datagram Protocol. Tiedonsiirtoprotokolla reaaliaikaisen datan siirtoon.
OBT	Kiinalainen OBTPA-yritys, joiden kaiutinjärjestelmään työni perustuu.
Käyttäjäohjelma	On sana, jota käytän viittaamaan ohjelmaan, jolla käyttäjä kommunikoi muiden käyttäjäohjelmien ja palvelimen kanssa.
VoIP	Voice over IP. Protokolla jolla siirretään puhetta verkon yli.
P2P	Peer to Peer. Vertaisverkko jossa käyttäjät toimivat palvelimina ja asiakkaina samaan aikaan.
Pakkauksenhallitsija	Pakkaa ja purkaa datavirrat tai signaalit siirtoa ja tallennusta varten. Englanniksi codec.
LGPL	GNU Lesser General Public License. On vapaan lähdekoodin lisenssi, joka ei vaadi käyttäjää julkaisemaan koodia käytettäessä.

1 Johdanto

Työni aiheena on lähiverkossa toimivan kuulutusjärjestelmän käyttäjäohjelman luonti. Kuulutusjärjestelmiä käytetään muun muassa kouluissa, sairaaloissa ja virastoissa ilmoittamaan yleisistä asioista koko rakennukseen tai vain yhdelle osastolle. Kuulutukset voivat olla yksittäisiä ilmoituksia, jotka koskevat esimerkiksi auton siirtämistä, tai ne voivat olla myös ajastettuja tapahtumia kuten koulujen välituntikellot tai suoraan toiselta laitteelta tulevia hälytyksiä esimerkiksi palohälytys.

Tarkoitukseni on luoda ohjain, jonka avulla nämä toiminnot olisi mahdollisia suorittaa verkon yli valmiin palvelimen kautta. Ohjain olisi siis ohjelmiston osa, joka toimii tietoliikennerajapintana ohjelmiston ja palvelimen välillä. Tämä valmis palvelin osaa keskustella siihen liitettyjen ja samasta verkosta löytyvien IP-kaiuttimien kanssa, sekä mahdollistaa äänen suoratoiston kaiuttimista, ajastettujen tapahtumien luonnin ja toistamisen. Kommunikaatio ohjaimen ja palvelimen välillä toteutetaan TCP-tietovirran avulla, mihin palvelin tarjosi valmista porttia. Suoratoisto käyttöliittymästä kaiuttimiin taas onnistuisi UDP- tietovirran kautta.

Työn tavoitteena on toteuttaa ohjain, joka luo helppokäyttöiset rajapinnat OBT-kuulutusjärjestelmän (OBT on kiinalainen yritys, joiden laitteisiin ohjelma perustuu). käyttäjäohjelmaan. Tärkeää on tehdä ohjain sellaisella tavalla, että sitä voitaisiin käyttää luotaessa käyttöliittymää useammalle alustalle ja se toimisi täysin erillisenä ohjelmana käyttäjäohjelman käyttöliittymästä. Tarkoitus on kuitenkin luoda lisäksi ohjelma, joka toteuttaa nämä rajapinnat työpöytäsovelluksena.

Työn toisena tavoitteena on selvittää ja esitellä kommunikaatiossa käytettyjen tietovirtojen toimintaa sekä suoratoiston tekniikoita. Tarkoituksena on ottaa selvää jo olevista tekniikoista sekä miten ja missä niitä käytetään ja kuinka itse voisin niitä työssäni käyttää. Erityisesti tarkoitus on keskittyä tietovirtoihin ja suoratoistoon Javassa. Tietovirroissa keskityn kertomaan niiden käytöstä verkkoliikenteessä. Suoratoistossa taas keskityn sen käyttöön median siirrossa.

Tein työn Avack Oy:lle. Avack on tamperelainen viestintäratkaisutalo, joka muun muassa suunnittelee hälytys- ja viestintäjärjestelmiä. Kuulutusjärjestelmän käyttäjäohjelman ohjaimen tekeminen valikoitui insinöörityöksi sen etänätekomahdollisuuden ja oman ohjelmointikokemuksen takia. Ohjaimelle oli tarvetta, sillä vaikka kuulutusjärjestelmään oli käyttöliittymä niin palvelimelle kuin käyttäjälle, eivät ne tarjonneet kaikkia haluttuja toimintoja. Yksi iso syy, miksi ohjainta tarvittiin, oli sen käyttäminen käyttäjäsovelluksen luomiseen puhelimeen.

2 Suoratoisto ja tietovirrat

Tässä luvussa tarkastellaan suoratoistoa ja tietovirtoja yleisesti sekä niihin liittyviä tekniikoita ja niiden soveltuvuutta kuulutusjärjestelmän ohjaimen toteutuksessa. Suoratoisto ja tietovirrat ovat tiedonsiirrontekniikoita, joissa dataa lähetetään kohteelle tai kohteille ja se on heti niiden käytettävissä. Etenkin suoratoisto on hyvin ajankohtainen, sillä nykyaikainen verkkoteknologia on mahdollistanut sen nousun hallitsevaksi viihdemedian kulutustavaksi.

Suoratoisto tai yleensä puhekielessä striimaus on tiedonsiirrontekniikka, jossa yleensä mediasisältö kuten ääni tai/ ja kuva toistetaan heti vastaanottajalle datan saapuessa hänen päätteelleen. Mediaa ei tallenneta päätteelle kokonaisuudessaan missään vaiheessa. Datan saapuessa se tallennetaan väliaikaiseen puskuriin, josta se muutetaan haluamaksemme mediasisällöksi. Esimerkkinä voi mainita Youtuben, jossa videon avatessa alkaa video lähes saman tien toistumaan. Samalla Youtube puskuroi videota muutaman sekunnin eteenpäin, jolloin datakatkoksen sattuessa ei video välttämättä katkea. Lopuksi viimeistään videosta poistuessa poistetaan se väliaikaisesta muistista.

Tietotovirtatapa siirtää dataa, jossa datalla on selvä lähtö- ja päätepiste. Tietovirtoja tarvitaan, kun halutaan saada dataa jostain laitteesta tai ohjelmasta, mutta laitetta tai ohjelmaa ei voida lukea koko aikaa. Tällöin laitteen tai ohjelman data jää tietovirran puskuriin, josta sitä kaipaava ohjelma voi sen lukea sitä tarvitessaan. Tietovirtojen käytön huomaa parhaiten tietokoneen hidastellessa tai käytettäessä raskaampia ohjelmia. Esimerkkinä avatessasi jonkun raskaamman ohjelman, joka vaati sisäänkirjautumista, alkaa tietovirta lukea näppäimistön syötteitä jo puskuriin ohjelman vielä avautuessa. Kun sisäänkirjau-

tumiskentät tulevat näkyviin, löytyvät niistä puskurista haetut syötteet. Tietovirtoja käytetään tietysti myös muissa ohjelmissa, mutta niissä suoritukset tapahtuvat niin nopeasti, ettei viivettä välttämättä huomaa.

2.1 Suoratoiston ja tietovirtojen toiminta

Tietovirran toimintaan kuuluu lukea dataa annetusta lähteestä sen oikeassa järjestyksessä tietokoneen välimuistiin puiskuriin ja kirjoittaa sama data samassa järjestyksessä toisaalle, esimerkiksi tiedostoon. Tarkoitus on siis siirtää dataa muuttumattomana kohteelta toiselle nopeasti. Siirron osapuolten ei tarvitse välttämättä käsitellä tiedostoa kokonaisuudessaan missään vaiheessa, vaan riittää, että käsitellään kulloinkin siirron kohteena oleva osa. Eli hyvin samanlainen idea kuin suoratoistolla. Suoratoiston voikin toteuttaa kokonaan tietovirroilla, mikäli käyttää TCP-protokollaa, mutta toteutus voi olla hyvin raskas verkkoliikenteelle.

Tietovirroilla on kuitenkin iso tehtävä kaikkien suoratoistoratkaisujen toiminnassa. Kun aletaan toteuttaa suoratoistoa, ei koko tiedostoa voi lähettää kerralla. Tietovirtojen avulla voidaan lukea kätevästi esimerkiksi toistettavaa tiedostoa, digitaalista äänivirtaa mikrofonista tai käyttäjän antamia syötteitä. Samasta tietovirrasta voidaan helposti kirjoittaa luetut arvot halutun kokoisiin paketteihin suoratoistoa varten. Vastaavasti vastaanottavassa päässä uusi tietovirta voi lukea saapuneen paketin ja kirjoittaa sen esimerkiksi tiedostoon tai äänen muodossa kaiuttimille.

Kuten tietovirrat myös verkkoteknologiat ovat iso osa suoratoiston toimintaa. Merkittävä valinta suoratoiston verkkoliikenteen toteutuksessa on, käytetäänkö tiedonsiirron UDP- vai TCP-protokollaa. TCP on yleisin tietoliikenneprotokolla, joka lähettää tavujonoja kahden pisteen väillä. TCP-protokollan etuja ovat sen ominaisuudet varmistaa pakettien saapuminen perille, kuten myös pakettien oikean järjestyksen varmistaminen muun muassa. Pakettien saapumista perille TCP-protokolla valvoo vaatimalla ACK-vastauspaketteja jokaisen paketin lähetyksen jälkeen. Jos vastausta ei tule, lähetetään paketti uudestaan. Pakettien oikeaa järjestystä taas tarkkaillaan järjestysnumeroilla. UDP-protokolla on myös tietoliikenneprotokolla, mutta kevyempi ja nopeampi kuin TCP. UDP-protokolla ei tarkista saapuvatko paketit perille tai ovatko ne oikeassa järjestyksessä. Tämän takia vastaanottajan ei tarvitse lähettää mitään takaisin tai lähettäjän odottaa vastausta. Etuna

tässä on yleensä hyvin reaaliaikainen tiedonsiirto. Muita etuja verrattuna TCP-protokollaan on mahdollisuus moni- ja yleislähetyksiin.

Yleensä kuitenkin käytetään kummastakin protokollasta jatkettuja ja kumpaakin protokollaa käyttäviä protokollia kuten Skypen omaa Skype-protokollaa. Skype-protokolla käyttää UDP-protokollaa äänen siirtämiseen ja TCP-protokollaa muuhun tietoliikenteeseen [1.]. Jos puhutaan pelkästään TCP- ja UDP-protokollista, niin UDP on yleensä ollut suoratoistossa käytetty protokolla. Se on sitä vieläkin, mikäli suoratoissa toistettava data luodaan reaaliajassa, kuten VoIP-ohjelmissa eli verkon yli puhetta siirtävissä ohjelmissa esimerkkinä Skype tai reaaliaikaisissa verkkopeleissä, kuten monet ammuntapelit. TCP on taas nousut tallennetun median kuten äänen ja videon suoratoistossa käytetyksi protokollaksi. Tämä johtunee nykyisten datansiirtonopeuksien tarjoamasta mahdollisuudesta käyttää TCP-protokollaa tuomaan hyvätasoista mediaa riittävällä nopeudella.

UDP-ongelmat datan siirrossa korostuvat liikaa, kun kilpaillaan kuvan ja äänen laadussa, kuten musiikin ja elokuvien kohdalla. Mutta esimerkiksi mikrofonista ääntä toistettaessa, jos paketti katoaa se ei haittaa, sillä uusi nauhoitettu ääni on jo korvannut sen. Saman periaatteen takia UDP-protokolla sopii hyvin kaikkien reaaliajassa nauhoitettujen tietojen suoratoistoon. Ehkä isoin syy, miksi UDP-protokollan käyttöä tarvitaan vielä nykyisillä verkkonopeuksilla, on sen mahdollisuus ryhmä- ja yleislähetykseen, joita TCP-protokollassa ei ole, kuten jo aiemmin todettiin. Tämä on tarpeellista juurikin kuulutusjärjestelmissä ja VoIP-ohjelmissa, joissa vastaanottavia kohteita voi olla monia.

TCP-protokolla on aina ollut luotettavampi, mutta raskaampi vaihtoehto median siirtämiseen kuin UDP. Tämänhetkisillä datansiirtonopeuksilla ja kehittyneillä välimuistin tekniikoilla palvelimen ja käyttäjän välistä kuormaa saadaan pienennettyä. Kolmas kuormaa pienentävä ja nousussa oleva tekniikka on P2P eli vertaisverkkoja hyväksi käyttävä tekniikka. P2P-tekniikkassa käyttäjät toimivat asiakkaina ja palvelimina samaan aikaan, jolloin varsinaisen palvelimen kuormaa saadaan pienennettyä. Spotifylla suurin osa datan siirrosta tapahtuu P2P-tekniikalla [2.]. Netflixillä on ollut jo muutaman vuoden P2P-asiantuntiaja haussa [3.]. Käytänkin tässä esimerkkinä, kuinka Spotify käyttää käyttäjien välimuistin ja P2P-tekniikka median suoratoistossa. Tuloksena saadaan suhteellisen hyvä kaista johtuen useista käyttäjistä, mutta samalla saadaan varmistettu data, koska protokollana toimii TCP. Kun Spotify alkaa etsimään seuraavaksi soitettavaa kappaletta,

etsii se ensiksi, olisiko kappale mahdollista saada P2P-tekniikalla. Ehtoina P2P jakamiseen kappaleilla on, että joko ne löytyvät ladattuina käyttölaitteen muistiin uloskirjautumistilaa varten tai kokonaan välimuistiin ladattuna kuuntelun takia [2.].

Myös pakkauksenhallinta tai yleisemmin codec on yksi tärkeimmistä osista toimivan ja hyvälaatuisen suoratoiston toteutusta. Pakkauksenhallinnan tärkeydestä äänen suoratoistossa etenkin VoIP-ohjelmissa kertovat enemmän Samrat Ganguly ja Sudeept Bhatnagar kirjassaan VoIP Wireless, P2P and New Enterprise Voice over IP, johon tässä viittaan [4.]. Perinteisimmillään pakkauksenhallinnan tarkoitus on vain muuttaa mikrofonista tai muusta lähteestä saatu ääni siirrettävään formaattiin. Pakkauksenhallinta voi tehdä kuitenkin enemmän. Sillä pystyy muun muassa korjaamaan verkkoliikenteen tuottamia häiriöitä ,kuten viivettä, pakettien katoamista tai värinää, jotka vaikuttaisivat äänenlaatuun. Näin toimii ainakin Skypen käyttämä GIPS-pakkauksenhallinta, joka on Global IP Sound -yrityksen luoma pakkauksen hallinta reaaliaikaiseen kommunikaation [5.]. Pakkauksen hallintaa valitessa olisi hyvä tuntea lähettävä- ja vastaanottavapää kunnolla. Eri pakkaushallintojen käyttö vaikuttaa äänen laatuun merkittävästi [6.].

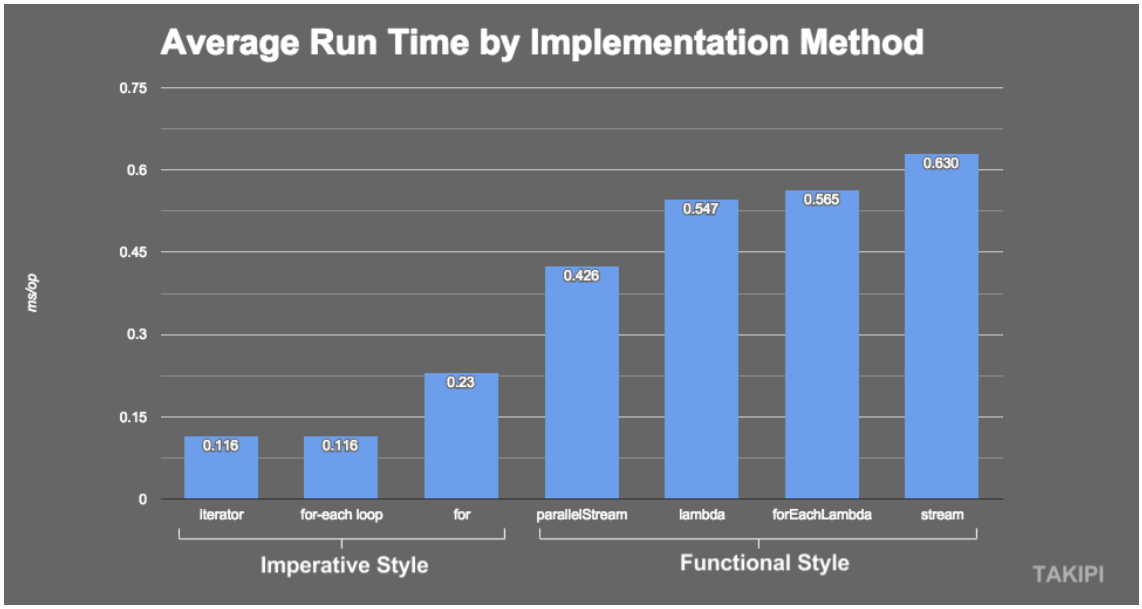
2.2 Suoratoisto ja tietovirrat Javassa

Suoratoiston toteuttaminen Javalla on siitä suhteellisen helppoa, että Java tarjoaa kaikki työkalut yksinkertaiseen suoratoiston mahdollistavaan käyttäjäohjelmaan ja palvelimen luontiin. Java.net-kirjasto mahdollistaa niin TCP-yhteyden kuin UDP-pistokkeen luonnin suoratoiston verkkoliikennettä varten. Vuorostaan taas javax.sound.sampled-kirjasto tarjoaa työkalut mikrofonin ja ääniaalto tiedostojen lukemiseen ja pakkaamiseen. Omassa ohjelmassani en tarvinnut videon käsittelyä, mutta nopea tutkimuksen perusteella moni suositteli käyttämään siihen FFmpeg-pakkauksenhallintaa, joka on LGPL-lisenssin alainen monen alustan monimediakehys. LGPL-lisenssi on vapaan lähdekoodin lisenssi, mutta mahdollistaa käytön ilman käytetyn koodin julkaisemista [7.]. FFmpeg sopii myös tarvittaessa äänen käsittelyyn [8.].

Javan mukana tulevat työkalut mahdollistavat kuitenkin vain yksinkertaisen suoratoiston ja ovat hyvin alttiita häiriöille. Javassa muun muassa ei ole omaa kirjastoa RTP eli reaaliaikaisen siirtoprotokollan tai RTSP eli reaaliaikaisen suoratoistoprotokollan hyödyntämiseksi. JMF- eli Java Media -kehys tarjosi aikanaan RTP-protokollaa Javaan, mutta

tämä on jo vanhentunut. Tarjolla on onneksi kolmannen osapuolen tuotteita kumpaakin protokollaa varten kuten efflux [9.]. RTP tai RTSP eivät kuitenkaan ole välttämättömiä, mutta koska UDP ei itsessään määritä kuin pituuden ja kohdeportin sekä joskus lähtöportin ja tarkistussumman, voi saapuva data toistua virheellisesti. Esimerkiksi yhteyden hidastuessa ei vastaanottaja tiedä sitä, sillä aikaleimat puuttuvat ja ääni toistuu virheellisesti.

Yrittäessäni etsiä tietoa Javan tietovirroista joko niiden hyvistä tai huonoista puolista tai verrattuna muihin ohjelmointikieliin. Suurin osa kritiikistä osui Java 8 lisättyihin lambda- ja rinnakkaistietovirtoihin. Kuten kuvasta 1 näkee, tai kuten Alex Zhitnitsky blogissaan esittää, ovat ne selvästi perinteisiä iteraattoreita ja for-each-loop-rakenteita hitaampia [10.]. Java 8 tarjoamat virrat on tosin suunniteltu funktionaalisen ohjelmoinnin mahdollistamiseen Javalla ja siksi eroavat Javan perinteisistä tietovirroista kuten InputStream.



Kuva 1: Kuinka nopeasti operaatiot löysivät 100 000 satunnaisesta numerosta taulukkolistasta [10.].

2.2.1 Palvelin ja käyttäjä kommunikaatio

Palvelimen ja käyttäjän kommunikaatio tai vastaavasti kahden käyttäjän kommunikaatio on helppo toteuttaa Javalla käyttäen TCP-pistokkeita yhteyden luontiin. Yleensä palvelin avaa aluksi pistokkeensa, jolla on portti ja IP-osoite. Jos pistoke on ennalta määritelty,

voivat käyttäjät ottaa siihen automaattisesti yhteyttä. Muita vaihtoehtoja on esimerkiksi määrittää kysely komento, jotta kun käyttäjä haluaa yhdistää palvelimeen, voi se lähettää yleislähetyksellä kyselyn, jossa se kertoo oman IP-osoitteensa ja porttinsa, jotta palvelin voi luoda taas yhteyden käyttäen TCP-protokollaa. Jotta yleislähetyksellä huhuilu toimisi täytyy palvelimen kuunnella verkon UDP lähetyksiä. Yleislähetystä käytettäessä tarvitsee käyttäjän olla joko samassa verkossa tai tietää, missä verkossa palvelin on. Kummassakin tapauksessa on viestit hyvä salata. Kun kaksi käyttäjää haluaa avata kommunikaation, voivat he pyytää mahdollisesti palvelinta välittämään tarvittavat tiedot tai käyttää yleislähetystä huhuilemaan toista käyttäjää verkosta.

Kun yhteys käyttäjän ja palvelimen tai käyttäjien välillä on luotu, voivat he käyttää TCP-pistokkeesta saatavia tietovirtoja keskusteluun. Palvelimen ja käyttäjien väillä käytävä keskustelu riippuu suoratoistojärjestelmän tarkoituksesta. Esimerkiksi VoIP-ohjelmissakin voi olla hyvin paljon eroja. Ohjelma, joka toimii kuten Skype, käyttää yhteyttä palvelimeen lähinnä käyttäjätietokannan hakuihin, kuten kirjautumiseen, oman profiilin muokkaamiseen tai muiden käyttäjien hakemiseen. Muita komentoja voi olla puhelu pyynnön lähettäminen joko palvelimen kautta tai suoraan tutulle palvelimelta saatujen tietojen kautta. Toisenlaisessa VoIP-ohjelmassa kuten TeamSpeakissä, jossa ei erillisiä käyttäjiä edes tarvita vaan käyttäjät voidaan luoda dynaamisesti IP-osoitteen perusteella niiden yhdistäessä. Tällaisessa ohjelmassa palvelin yhteyttä käytetään liikkumaan palvelimen eri kanavien välillä, jotka yhdistävät dynaamisen käyttäjän kanavan ryhmälähetykseen. Ohjelmassa kuten Netflix taas tarvitaan palvelinyhteyttä käyttäjätietokantaan, videotietokantaan sekä itse videon suoratoistoon.

Palvelin- ja käyttäjäyhteyden toteuttaminen käyttäen Javan TCP-pistokkeita oli suoratoisto-ohjelmiston helpoimmin toteutettava osuus. Itse en törmännyt omassa työssäni TCP-pistokkeissa mihinkään teknisiin ongelmiin. En myöskään etsiessäni tietoa löytänyt kritiikkiä pistokkeiden toiminnasta.

2.2.2 Tiedostojen suoratoisto

Seuravaksi kerron, mitä työkaluja Java tarjoa tiedostojen suoratoistoon, erityisesti videon ja äänen suoratoistoon. Kuten aliluvun 2.2 alussa mainitsin, ovat Javan omat työkalut tältä osalta hyvin vähäiset, mutta hyviä kolmannen osapuolen kirjastoja on tarjolla.

Aliluvun 2.2 alussa esittelin Javan oman javax.sound.sampled -kirjaston äänen pakkauksen hallintaan. Kirjasto on kuitenkin äänitiedostojen käsittelyyn hyvin vajavainen jo siinä, ettei se esimerkiksi hallitse mp3-tiedostoja ollenkaan. Äänitiedostoformaatit, jotka javax.sound.sampled hallitsee, ovat WAVE, AIFF, AIFC, SND ja AU. Nämä formaatit on selitetty tarkemmin esimerkiksi Tim Fisherin artikkelissa [11.]. WAVE on tavallisesti Windowseissa käytetty CD-tasoinen pakkaamaton äänitiedosto. AIFF on Applen kehittämä WAVE-ääntä vastaava pakkaamaton formaatti. AIFC on AIFF-pakattu versio. SND on myös Applen kehittämä formaatti, joka voi sisältää joko yhden äänitteen tai useampia äänen ohjauskomentoja. AU taas on Sun Microsystemsin käyttöjärjestelmissään käyttämä ääni [12.].

Mikään yllä mainituista ei kuitenkaan ole yhtä suositeltava äänen suoratoistoformaatti kuin esimerkiksi Ogg Vorbis. Ogg Vorbis on täysin avoimen lähdekoodin patenttivapaa pakkausmuoto. Ogg vorbis -formaatti suunniteltiin korvaamaan mp3, kun mp3:n lisensoimisestä kerrottiin [13.]. Ogg Vorbis -formaattia käyttää muun muassa Spotifyn äänen suoratoistoon [14.]. Koska Java ei itsessään mahdollista Ogg-muotoisten tiedostojen pakkaustenhallintaa, tarvitaan sitä varten toinen työkalu kuten aikaisemmin mainittu FFmpeg.

Aliluvussa 2.2 mainitsin, ettei Java sisällä työkaluja videoiden käsittelyyn. Kerroin myös, kuinka FFmpeg käy hyvin videoiden pakkauksen hallinnaksi. Itse datan suoratoisto ei muutu, mutta vastaanottavassa päässä tarvitaan kirjasto, jolla vastaan otettu video voidaan toistaa. Java tarjoa tähän JavaFX-mediakirjaston, jolla pystyy toistamaan FLV- ja MPEG-4-tiedostoja [15.]. Muita vaihtoehtoja on esimerkiksi vlcj, joka toimii usealla alustalla ja tukee useampaa formaattia. Vlcj on kuitenkin GPL-lisenssin alainen, joka vaati käytettäessä lähdekoodin julkaisua. Vaihtoehtoisesti vlcj tarjoaa myös kaupallista lisenssiä.

Kuten aliluvussa 2.1 on perusteltu, on TCP parempi kuin UDP tiedostojen suoratoistoon. Itse toteutin projektissani äänitiedostojen toiston käyttäen UDP-protokollaa ja WAVE-formaattia. UDP-protokollaa tarvittiin yleislähetystä varten, ja WAVE oli ainoa javax.sound.sampled-kirjaston tarjoamista formaateista, jota käyttämäni OBT IP -kaiuttimet pystyivät järkevän kuuloisesti toistamaan. Äänen laatu ja ymmärrettävyys oli tällöin siedettävä, kun toimittiin vain puhetta sisältävillä tiedostoilla. Testasin myös toistaa mu-

siikkia näin, mutta tulos kuulosti hirveältä. IP-kaiuttimet, jotka vastaanottivat datan, olisivat tarvinneet, jonkun näköisen puskurin sekä lähetetty data olisi tarvinnut aikaleiman kuten RTP-protokollassa pitämään äänen tahdissa. Näihin en voinut kuitenkaan vaikuttaa sillä kaiuttimien toiminta oli määritelty etukäteen. TCP suoratoistoa en kokeillut äänen kanssa, mutta sitäkin varten olisi tarvinnut puskurin vastaanottimeen ja aikaleimat lähetykseen pitämään tahtia.

2.2.3 Äänen suoratoisto

Seuraavaksi kerron reaaliaikaisen äänen suoratoistosta. Reaaliaikaisen äänensuoratoisto tuo joitain lisä haasteita verrattaessa nauhoitetun äänen suoratoistoon. Näitä haasteita ovat muun muassa äänen nauhoittaminen sekä latenssin minimoinnin tärkeyden lisääntyminen. Helpottavana tekijänä on kuitenkin, ettei reaaliaikaisen äänen tarvitse olla laadultaan yhtä tarkkaa kuin tallennetun äänen. Tämä johtuu siitä, että reaaliaikainen ääni on yleensä puhetta tai muuten yksinkertaisempaa ääntä kuin musiikki tai muu tallennettu ääni. Reaaliaikaisessa äänensuoratoistossa nopeus on myös tärkeämpää kuin laatu.

Reaaliaikaisen äänen suoratoistossa tuleekin ongelma käytettäessä Javan omaa java.sound API -kirjastoa. Tällä viittaan tutkimukseen, jossa Nicolas Juillerat, Stefan Müller Arisona ja Simon Schibiger-Banz olivat tutkineet Javaa reaaliaikaisessa äänen prosessoinnissa. Tutkimuksestaan he kertovat raportissaan Real-Time, Low Latency Audio Processing In Java [16.]. On hyvä huomata, että raportin testeissä käytettiin Javan 6 versiota tämänhetkisen version ollessa 8. Raportissa muun muassa kerrotaan java.sound API -kirjaston, jota itse käytin projektissani mahdollistavan parhaimmillaankin vain 150 ms latenssin, kun tavoite pienen latenssin prosessointiin olisi 20 ms. Raportissa he olivat ratkaisseet ongelman käyttämällä ALSA API -kirjastoa, joka on kehittynyt Linux-ääniarkkitehtuuri [17.]. Koska API on Linuxille, täytyy sen kutsut kääriä käyttäen JNI-rajapintaa eli Java Native Interfacea, joka mahdollistaa Javan virtuaalikoneen kutsut ohjelmista, jotka on tehty käyttäen muita ohjelmointikieliä kuten C:tä. Lopuksi tarvittiin vielä RtAudion API-kirjastoa auttamaan ALSA toimimaan muillakin alustoilla kuin Linuxilla. Yritin myös etsiä muita mahdillisia API-kirjastoja mikrofonin lukemiseen Javalla reaaliajassa, mutta en löytänyt.

Toinen mahdollinen ongelma, joka esitettiin yllämainitussa raportissa koskien Javaa ja reaaliaikaista suoratoistoa, koski Javan roskien keruuta. Tämä ei kuitenkaan heidän testeissään tällä kertaa asettunut ongelmaksi, mutta artikkelin tekijät mainitsivat, että se saattaa aiheuttaa suurtakin värinää johtuen virtuaalikoneen tarpeesta estää muiden säikeiden toimintaa roskien keruun ajaksi. [17.]

Kuten aikaisemmin jo mainitsin toteutin äänen nauhoittamisen käyttämällä java.sound API-kirjastoa. Itse en ole mitannut latenssia, mutta sen merkitys ei ole kauhean iso omassa projektissani, koska kyse on yksisuuntaisesta suoratoistosta. Suurin ongelma tällä hetkellä omassa toteutuksessa on reaaliaikaisen äänen suoratoistossa kasvava viive. Suoralähetyksen ollessa käynnissä kasvaa viive noin sekunnin minuutissa. Tähän ongelmaan en ole vielä löytänyt ratkaisua. En kuitenkaan usko sen johtuvan java.sound API -kirjastosta.

Tiedon siirron reaaliaikaisen äänen suoratoistossa olen toteuttanut käyttäen UDP-protokollaa, mikä on jo aikaisemmin todettu järkeväksi. Toteutuksessa käytän samaa menetelmää kuin äänitiedostoja lähettäessä. Erona on että tällä kertaa äänivirran lähteenä on tiedoston sijaan mikrofoni, josta luen kilotavun verran kerrallaan UDP-pakettiin lähetettäväksi. Lähetys toimii hyvin lukuun ottamatta ylempänä mainitsemaani ongelmaa. Tämäkään ei ole siinä mielessä merkittävä, mikäli lähettäjä ei pidä puheessaan pitkiä taukoja ei käyttäjä huomaa viivettä puheen virrassa.

3 OBT-kuulutusjärjestelmä

Tässä luvussa esittelen OBT-kuulutusjärjestelmää ja etenkin sen ohjainta. Kerron aluksi, mitä toiminnallisia vaatimuksia ohjaimen toteuttamissa oli. Vaatimusten pohjalta kerron toiminto kerrallaan, kuinka ne on toteutettu. Toteutusten kuvausten yhteydessä annan esimerkkejä ratkaisuissa käytettyjen tekniikoiden käytöstä. Jokaisen toiminnon toteutuksen yhteydessä kerron siihen liittyvistä erityisistä haasteista.

3.1 Ohjaimen toiminnan määrittely

Ohjaimen toimintojen määrittelyyn otin lähestymistavaksi luoda ohjaimen vaadittavista toiminnoista käyttäjäkertomuksia. Käyttäjäkertomuksia suunnittelussa käytin Avackilta

saatuja ohjeita, joita toimintoja ohjaimella pitäisi olla. Vertasin näitä ohjeita edeltävään OBT-käyttäjäohjelmaan ja OBT-palvelimen SDK-ohjainsarjaan.

1. Käyttäjän täytyy pystyä luomaan TCP-yhteys palvelimeen syöttämällä IP-osoite.

2. Käyttäjän täytyy pystyä kirjautumaan palvelimelle syöttämällä salasana ja käyttäjätunnus.

3. Käyttäjän täytyy pystyä seuraamaan järjestelmän laitteita ja tapahtumia oikeuksiensa mukaan.

4. Käyttäjän täytyy pystyä muuttamaan valitsemiensa kaiuttimien äänenvoimakkuutta.

5. Käyttäjän täytyy pystyä hakemaan palvelimelle ja kaiuttimiin tallennettuja kappaleita.

6. Käyttäjän täytyy pystyä toistamaan kaiuttimiin tallennettuja kappaleita haluamistaan kaiuttimista

7. Käyttäjän täytyy pystyä pysäyttämään kaiuttimiin tallennettujen kappaleiden toisto halutuissa kaiuttimissa.

8. Käyttäjän täytyy pystyä aloittamaan suoratoisto käyttäjälaitteeseen kiinnitetystä mikrofonista haluttuihin kaiuttimiin.

9. Käyttäjän täytyy pystyä lopettamaan suoratoisto käyttäjälaitteeseen kiinnitetystä mikrofonista haluttuihin kaiuttimiin.

10. Käyttäjän täytyy pystyä aloittamaan suoratoisto käyttäjälaitteesta löytyvästä WAVE-äänitiedostosta haluttuihin kaiuttimiin.

11. Käyttäjän täytyy pystyä lopettamaan suoratoisto käyttäjälaitteesta löytyvästä WAVE-äänitiedostosta haluttuihin kaiuttimiin.

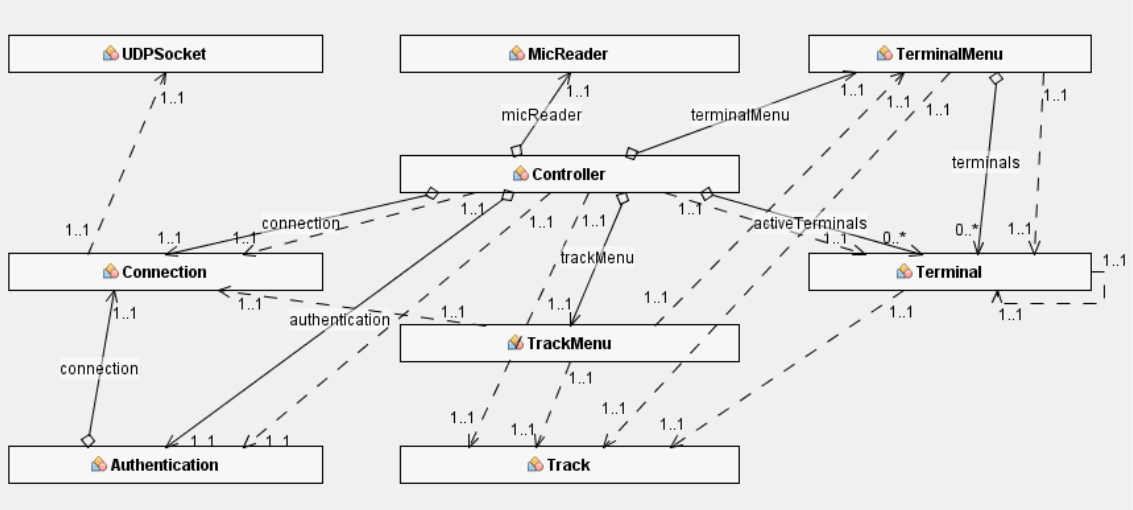
12. Käyttäjän täytyy pystyä äänittämään WAVE-äänitiedostoja.

3.2 Käyttöliittymän toiminnan määrittely

Toteutuksessa käyttöliittymän tarkoitus oli testata ohjaimen rajapintojen toimintaa. Siksi sen ulkoasu ei merkitse juurikaan. Tärkeintä käyttöliittymällä on toteuttaa kaikki alilu-vussa 3.1 määrittelyt ohjaimen toiminnot graafisesti.

3.3 Ohjaimen toteutus

Ohjaimen eri tehtävät on jaettu niiden toimintaa kuvaaviin luokkiin. Kaikkia muita luokkia ohjaa Controller-luokka, joka toteuttaa myös ohjaimen rajapinnan käyttäjälle. Loin koodin pohjalta ohjelmasta luokkamallin easyUML-liitännäisellä, jotta ohjaimen eri luokkien yh-teydet olisi helpompi hahmottaa [18.]. Kuten kuvasta 2 näkee, piilotin luokkamallin muut-tujat sekä metodit, jotta saisin sen mahtumaan kuvaan.



Kuva 2: Ohjaimen supistettu luokkamalli.

Kuvasta 2 puuttuu myös suoralähetyksestä vastaava Broadcast-luokka kokonaan. Tämä johtunee siitä, että Broadcast-olio luodaan ainoastaan, kun aloitetaan suoratoisto ja lisä-tään samalla Controllerin listaan, joka pitää sisällään kaikki senhetkiset suoratoistot. Broadcastin toiminnasta kerrotaan enemmän aliluvussa 3.3.3.

Koska Controller toteuttaa rajapinnan ohjainta käyttävälle ohjelmalle ja ohjaa muita ohjaimen luokkia, osaa se tarkistaa luokkien tilat ja tarvittaessa luoda luokat niitä kutsuttaessa. Näin varmistetaan, että käyttäjän ei tarvitse ohjelman käynnistyksessä luoda kaikkia ohjaimen luokkia, mutta ne löytyvät ohjaimelta, mikäli niitä tarvitaan ja vaadittavat syötteet on annettu.

Ohjaimessa kaikki luokat ovat riippuvaisia Connection-luokasta jollain tavalla. Connection-luokan vastuulla on TCP-yhteyden luonti palvelimeen sekä UDPSocket-olioiden luonti suoratoistoa varten. Näistä on enemmän aliluvussa 3.3.1. Connectionin lisäksi ovat muut luokat riippuvaisia myös Authentication-luokasta. Authentication-luokka vastaa palvelimelle kirjautumisesta. Ilman kirjautumista palvelin ei vastaa käyttäjäohjelman sille lähetettyihin komentoihin. Authentication-luokka vastaa myös palvelimelta haettavien perustietojen hausta. Viimeinen yleisesti vaadittu luokka on TerminalMenu ja samalla sen hallitsemat Terminal-oliot. TerminalMenu-luokka kääntää palvelimelta saadut tiedot kaiuttimista luettavaan muotoon ja tallentaa ne Terminal-olioita sisältävään taulukkolistaan. TerminalMenu-luokka vastaa muunnoksen lisäksi listan ylläpidosta, kuten mitkä kaiuttimet ovat milloinkin vapaana. Kaiuttimien äänenvoimakkuuden muuttaminen on myös TerminalMenu-luokan tehtävä.

Muita ohjaimen luokkia ovat MicReader, TrackMenu ja Track. MicReader vastaa mikrofonin lukemisesta. Aiemmin mainittu Broadcast-luokka tarvitsee MicReaderin suoratoistoa varten. Muina ominaisuuksina MicReaderia käytetään äänitiedostojen formaatin lukemiseen sekä omien äänitiedostojen nauhoittamiseen. MicReaderistä enemmän aliluvussa 3.3.5. Track on olio, jonka avulla saadaan listattua kätevästi palvelimelta ja kaiuttimista löytyvät kappaleet. Track-oliot tallennetaan palvelimen tapauksessa TrackMenu-luokkaan ja kaiuttimien kohdalla niiden kaiuttimia vastaavien Terminal-olioiden Track-taulukoihin. TrackMenu-luokan tehtävä on hakea palvelimen ja kaiuttimien kappaleet sekä aloittaa ja lopettaa kappaleiden toisto.

Rajapintana toimimisen lisäksi Controller luo myös palvelimen lähettämiä muutostietoja lukevan säikeen TerminalInfoListener. Säie luodaan TerminalMenu-luokan luonnin yhteydessä, ja se lukee TCP-pistoketta ja lähettää tiedon TerminalMenu-luokalle kaiuttimientilan päivittämistä varten.

3.3.1 Kommunikaation toteutus

Kommunikaation toteutuksessa käytin Javan TCP-pistoketta, josta sain sisään ja ulos menevät tietovirrat. Pistokkeen luomiseen tarvittiin IP-osoite ja portti. IP-osoite pyydettiin käyttäjältä ja portti oli määritelty luokassa, kuten koodiesimerkissä 1.

```
private Socket socket;
private DataOutputStream out;
private BufferedInputStream ins;
private int port = 1234;
private String ip;
private UDPSocket[] udpSockets = new UDPSocket[10];

// Creating connection for controller
public Connection(String ip) throws IOException {
    this.ip = ip;
    socket = new Socket(ip, port);
    out = new DataOutputStream(socket.getOutputStream());
    ins = new BufferedInputStream(socket.getInputStream());
    for (int i = 0; i < udpSockets.length; i++) {
        udpSockets[i] = new UDPSocket(ip);
    }
}
```

Koodiesimerkki 1. Yhteyden luonti

Connection-luokan luonnissa alustetut tietovirrat pyydetään muissa luokissa käyttöön käyttäen perinteisiä get-metodeja. Koodiesimerkistä 1 huomaamme myös, kuinka yhteyden luonnin yhteydessä alustamme UDPSocket-oliot valmiiksi suoralähetyksiä varten. UDPSocket-oliosta on lisätietoa aliluvussa 3.3.3.

Kun yhteys kommunikaatioon on luotu, loput kommunikaation toteuttamisesta on komento- ja luokkakohtaista. Komentojen lähettämisen ja vastaanottamisen suunnittelussa käytin OBT-palvelimen SDK-ohjaussarjaa, josta sain komentojen ID-numerot sekä komentojen sisällöt. Komennot kulkevat käyttäjäohjelman ja palvelimen välillä tavutaulukkoina, joten ohjaimen tehtävä on kääntää vastaanotetut tavut esitettävään muotoon, kuten lukujonoiksi ja numeroiksi. Komentoja palvelimelle lähetettäessä tulee taas muuttuvat syötteet, kuten valitut kaiuttimet tai käyttäjänimi muuttaa tavuiksi. Tavuiksi muutetut syötteet tallennetaan oikeille paikoille muun komennon vaatiman datan kanssa ja lähetetään palvelimelle. Jokaista komentoa kohti on luotu oma metodi, jonka tehtävä on annettujen syötteiden pohjalta luoda komennon tavutaulukko.

Tavumuunnoksia tehdessä on huomioitava alkuperäisen luvun suuruus, sillä yhdellä tavulla pystyy ilmoittamaan vain 256 eri numeroa. Javassa nämä on jaettu -128 ja 127 välille [19.]. Tavujen koosta johtuen joutuu käyttämään usein useampaa tavua numeron ilmoittamiseen.

3.3.2 Kirjautumisen toteutus

Kirjautumisen toteutus käyttää aliluvussa 3.3.1 esitettyä tapaa kommunikoida käyttäen tietovirtoja ja tavutaulukoita. Kun halutaan kirjautua, testataan aluksi, löytyykö yhteys palvelimeen. Jos yhteys löytyy, lähetetään käyttäjätunnus ja salasana palvelimelle. Vastauksena palvelin lähettää onnistumisen tai jonkin virheen, kuten väärä salasana tai käyttäjätunnus.

Kun on kirjauduttu sisään, haetaan kaikki käyttäjän tarvitsema tieto palvelimelta, kuten käyttäjän käytössä olevat kaiuttimet ja käyttäjäoikeudet. Kaiken oleellisen tiedon saavuttua voidaan luoda TerminalMenu-luokka. TerminalMenu-luokan luonti ei ole suoraan yhteydessä sisään kirjautumiseen, mutta se on oleellinen osa sitä, sillä se jatkaa palvelimen ja käyttäjäohjelman yhteyden ylläpitoa kirjautumisen jälkeen.

Kirjautumista palvelimelle ei ole suojattu millään tavalla, vaan tieto kulkee selkokielisenä tavuvirtana porttien välillä. Tähän on varmaan syynä, että uskotaan verkon, jossa kuulutusjärjestelmä toimii, hoitavan suojauksen.

3.3.3 Suoratoiston toteutus

Suoratoiston toteutuksessa käytin Javan UDP-pistokkeita datan lähettämiseen. Koska ohjelman tarkoitus oli kuuluttaminen, ei ohjelman tarvinnut ottaa vastaan suoratoistettua dataa. Kaikki ohjelman suoratoistama data löytyy tallennettuna käyttölaitteelta tai tulee käyttölaitteen oletus mikrofonista. Mikrofonin ja äänitiedostojen lukemiseen käytin javax.sound.sampled -kirjastoa.

Kun suoratoiston aloitusta kutsutaan Controller-luokassa, pitää sille antaa taulukkolistana kaiuttimet, mistä halutaan kuulutuksen tapahtuvat. Jos halutaan tehdä suoratoisto tiedostosta, pitää myös tiedoston nimi antaa. Ohjelma tarkistaa, että taulukolista ei ole olematon ja TerminalMenu-luokka on luotu. TerminalMenu-luokassa tarkistetaan, ovatko

halutut kaiuttimet vapaina ja palautetaan niistä vapaana olevat kaiuttimet. Seuravaksi luodaan MicReader-luokka, mikäli sitä ei ole vielä luotu, ja samalla tarkistetaan, käyttääkö joku muu ohjelman suoritus mikrofonia. Jos suoratoiston on tarkoitus tapahtua tiedostosta, tarkistetaan tässä kohdassa, löytyykö haluttu tiedosto. Kun kaikki tarkistukset on tehty, käynnistetään mikrofonin luku, jos mikrofonia tarvitaan. Jos mikrofonia ei tarvita, pyydetään suoraan UDP-pistoke Connection-luokalta.

Tässä kohtaa on hyvä esitellä, kuinka UDP-pistokkeet on luotu. Niiden luominen tapahtui Connection-luokan luonnin yhteydessä, kuten koodiesimerkistä 1 näkee. UDP-pistoke tarvitsee yleislähetystä varten käyttäjältä IP-osoitteen, kuten nähdään koodiesimerkistä 2.

```
private DatagramSocket udpSocket;
private InetAddress bCast;

public UDPSocket(String ip){
        try{
            udpSocket = new DatagramSocket();
            udpSocket.setBroadcast(true);
            bCast = InetAddress.getByName(ip);
        }catch(Exception e){
            System.out.println(e);
        }
    }
```

Koodiesimerkki 2. UDP-pistokkeen luonti

Koodiesimerkissä 2 luodaan aluksi UDP-pistoke ensimmäiseen vapaaseen porttiin. Tämän jälkeen asetamme pistokkeen mahdollistamaan yleislähetyksiä. Lopuksi asetamme käyttäjältä saadun IP-osoitteen yleislähetyksen IP-osoitteeksi. Riippuen millaisessa verkossa suoratoiston yleislähetys on tarkoitus suorittaa, voidaan tässä kohdassa käyttää myös esimerkiksi paikallista yleislähetysosoitetta 255.255.255.255.

Kun vapaa UDP-pistoke on saatu, voidaan luoda uusi Broadcast-olio. Broadcast-olio tarvitsee luontinsa yhteydessä lähetyksen kaiuttimet, kutsuvan käyttäjän käyttäjänimen, lähetyksen näytteenottotaajuuden, UDP-pistokkeen ja Connection-luokan kommunikaatiota varten. Tietoja Broadcast-olio käyttää lähettämään palvelimelle kaiuttimien ID-numerot, käyttäjänimen, UDP-pistokkeen portin ja lähetyksen näytteenottotaajuuden. Tietojen lähetys tapahtuu aliluvussa 3.3.1 esitetyllä tavalla. Kaiuttimet-olio tallentaa myös

sen paikalliseen taulukkolistaan. Palvelin välittää käyttäjältä sille annetut tiedot kaiuttimille, jotka osaavat tämän jälkeen hakea suoratoistettavaa dataa niille kerrotusta portista.

Broadcast-olion luonnin jälkeen pyydetään sitä aloittamaan suoratoisto. Suoratoistoa kutsuttaessa annetaan sille joko MicReader-luokka, jos toisto tapahtuu mikrofonista tai tiedosto, jos toisto tapahtuu tiedostosta. Kun suoratoistokutsu on laitettu Broadcast-oliolle, lisätään olio senhetkisistä Broadcast-olioista kirjaa pitävään taulukkolistaan.

Broadcast-olio suorittaa suoratoiston säikeessä lukemalla sille annetun MicReader-luokan luomaa tavuvirtaa tai tiedostosta luettaessa tiedostoa. Keskityn tässä, kuinka toteutin suoratoiston MicReader-luokan tavuvirrasta. Aliluvussa 3.3.4 kerron enemmän, kuinka suoratoistoa sovellettiin tiedostosta luettaessa ja aliluvussa 3.3.5 kerron, kuinka toteutin MicReader-luokan mikrofonin lukemisen. Kun Broadcast-olion suoratoiston suorittamista kutsutaan, asettaa se aluksi sille annetun MicReader-luokan paikalliseksi MicReader-luokan ilmentymäksi ja testaa, nauhoittaako mikrofoni. Tämän jälkeen siirrytään suoratoistavaan säikeeseen. Säikeessä alustetaan lukumäärää ylläpitävä kokonaisluku, tavutaulukon puskuri ja haetaan UDPSocket-oliosta UDP-pistoke. Seuraavaksi siirrytään toistorakenteeseen, joka jatkuu, kunnes lähetystä ohjaava totuusarvo muutetaan epätodeksi, kuten koodiesimerkistä 3 huomataan.

```
int read;
byte[] buff = new byte[1024];
DatagramSocket d = udpSocket.getDatagramSocket();
while (playBroadcast) {
    ByteArrayOutputStream out = micRe.getOut();
    byte audioData[] = out.toByteArray();
    // Reseting ByteArrayOutputStream to keep track what is already read
    micRe.resetOut();
    // Checking if we have data to send
    if (audioData.length > 0) {
        // Puting our audio byte data to AudioInputStream
        in = new ByteArrayInputStream(audioData);
        tempStream = new AudioInputStream(in, micRe.getBroadcastFormat(), audioData.length);
        // Splitting our data from AudioInputStream to smaller packets to send through udp
        while ((read = tempStream.read(buff, 0, buff.length)) > 0) {
            DatagramPacket packet = new DatagramPacket(buff, buff.length
                , udpSocket.getInetAddress(), udpSocket.getPort());
            packet.setData(buff);
            try {
                d.send(packet);
            } catch (Exception e) {
                System.out.println(e);
            }
        }
        // Closing the temorary streams
        tempStream.close();
        in.close();
    }

}
```

Koodiesimerkki 3. Suoratoisto mikrofonin tavuvirrasta.

Toistorakenteessa luetaan ensimmäiseksi MicReader-luokan ulostuleva tavutaulukovirta paikalliseen ulostulevaan tavutaulukkovirtaan, ja paikallinen virta muutetaan tavutaulukoksi. Tämän jälkeen kerrotaan MicReader-luokalle tyhjentää ulostuleva tavutaulukkovirta vanhasta datasta. Jos tavutaulukosta löytyy dataa, siirretään ne uuteen sisääntulevaan tavutaulukkovirtaan. Tästä virrasta voimme luoda sisääntulevan äänivirran. Äänivirtaa luetaan toistorakenteessa aikaisemmin määriteltyyn puskuriin, kunnes äänivirta on kokonaan luettu ja lähetetty. Puskurista saamme UDP-paketin data sisällön. UDP-paketille annamme myös sen määränpää osoitteen ja portin. UDP-paketti voidaan lähettää, kun kaikki tarvittavat tiedot ja data on asetettu. Kaiken datan lähdettyä sisääntulevasta äänivirrasta voidaan väliaikaiset virrat sulkea.

Suoratoisto loppuu, kun kutsutaan suoratoiston pysäytysmetodia. On myös mahdollista pysäyttää suoratoisto vain osassa suoratoistavista kaiuttimista. Jos suoratoiston pysäyttää vain osassa kaiuttimista pysähtyy suoratoisto vasta, kun se on pysäytetty kaikissa sille alussa määrätyistä kaiuttimista. Meneillään olevaan suoratoistoon ei voi lisätä kaiuttimia. Kun komento suoralähetyksen pysäyttämiseksi annetaan, tarvitsee käyttäjän an-

taa kaiuttimet, joissa suoratoisto lopetetaan. Metodi poistaa tämän jälkeen kaikki Broadcast-oliot niitä ylläpitävästä taulukkolistasta, jos Broadcast-olio ei sisällä yhtään lähettävää kaiutinta. Seuraavaksi metodi tarkistaa, löytyvätkö käyttäjän antamat kaiuttimet mistään lähettävästä Broadcast-oliosta. Jos annetut kaiuttimet löytyvät joltain Broadcast-oliolta, annetaan oliolle pysäytäsuoratoisto-käsky näissä kaiuttimissa. Kun suoratoisto kaiuttimissa on pysäytetty, tarkistetaan, onko Broadcast-oliolla yhtään toistavaa kaiutinta, jos ei poistetaan se niitä ylläpistävästä taulukkolistasta.

Kun suoratoistonpysäytys-pyyntö laitetaan Broadcast-oliolle, tarkistaa se aluksi, onko suoratoisto edes käynnissä. Jos suoratoisto on käynnissä, käyttää suoratoiston pysäyttäjä metodi aliluvussa 3.3.1 esitettyä kommunikointitapaa lähettämään palvelimelle kaiuttimien ID-numerot ja käyttäjänimen. Seuraavaksi metodi poistaa kaiuttimet tämän olion kaiuttimista listaa pitävästä taulukkolistasta. Jos kaiuttimien taulukkolista on tämän jälkeen tyhjä, pysäyttää se suoratoiston toistorakenteet muutamalle niitä ohjaavat totuusarvot epätodeksi ja vapauttaa sille annetun UDP-pistokeen.

3.3.4 Tiedostojen suoratoiston toteutus

Tiedostojen suoratoisto toimii hyvin samalla tavalle kuin suoratoisto, joka esiteltiin aliluvussa 3.3.3. Erona mikrofonista suoratoistamiselle on, että metodille annetaan tiedosto luettavaksi ulostulevan tavutaulukkovirran sijaan. Kun suoratoisto tapahtuu tiedostosta, osaa metodi myös itse pysäyttää suoralähetyksen, jos tiedosto päättyy, kuten huomaamme koodiesimerkistä 4.

```
int read;
// Creating AudioInputStream from file if it is not null
if (soundFile != null) {
    try {
        tempStream = AudioSystem.getAudioInputStream(soundFile);
        AudioFormat format = tempStream.getFormat();
    } catch (UnsupportedAudioFileException | IOException ex) {
        System.out.println(e);
    }
}
byte[] buff = new byte[1024];
DatagramSocket d = udpSocket.getDatagramSocket();
playSoundFile = true;
// Reading the file to byte array
try {
    while ((read = tempStream.read(buff, 0, buff.length)) > 0 && playSoundFile) {
        // Creating Datagram packet
        DatagramPacket packet = new DatagramPacket(buff, buff.length
            , udpSocket.getInetAddress(), udpSocket.getPort());
        packet.setData(buff);
        // Sending DatagramPacket
        try {
            d.send(packet);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    tempStream.close();
    stopBroadcast();
} catch (Exception e) {
    System.out.println(e);
}
```

Koodiesimerkki 4. Tiedoston suoratoisto

Kun Broadcast-oliota pyydetään aloittamaan tiedoston suoratoisto, tallentaa se aluksi sille annetun tiedoston paikalliseksi ja siirtyy tämän jälkeen säikeeseen toimittamaan suoratoistoa. Aluksi säikeessä tarkistetaan tiedosto, ja jos se on toimiva, luodaan siitä sisääntuleva äänivirta. Äänivirrasta saamme myös tiedoston formaatin. Seuraavaksi alustetaan UDP-pakettien puskuri ja UDP-pistoke sekä asetetaan toistorakennetta hallitseva totuusarvo todeksi. Kun alustukset on tehty, siirrytään toistorakenteeseen. Toistorakenne jatkuu, kunnes käyttäjä pysäyttää suoratoiston kaikissa tämän Broadcast-olion käyttämissä kaiuttimissa tai tiedosto loppuu. Toistorakenteessa datan lukeminen sisääntulevasta äänivirrasta puskuriin ja sen asettaminen paketteihin sekä pakettien lähettäminen toimii samalla tavalla kuin suoralähetyksessä mikrofonista. Kun tiedosto loppuu tai käyttäjä on sammuttanut kaikki Broadcast-olion kaiuttimet, sulkee säie väliaikaiset virrat ja lähettää suoralähetyksen lopetuskutsun.

3.3.5 Mikrofonin lukemisen toteutus

MicReader on ohjelman luokka, joka vastaa mikrofonin lukemisesta. Kun MicReader-luokkaa luodaan ensimmäistä kertaa, alustetaan mikrofoni vapaaksi, ulostuleva tavutaulukkovirta suoralähetystä varten ja suoralähetys sekä äänitysääniformaatit. Ääniformaatteja alustaessa niille annetaan näytteenottotaajuus, näytteen koko bitteinä, äänikanavien määrä, onko formaatti kirjattu sekä onko formaatin tavujärjestys merkitsevimmästä bitistä vai ei. Tästä kertoo koodiesimerkki 5.

```
try{
    this.broadcastFormat = new AudioFormat(32000.0f, 16, 1, true, false);
    this.recordingFormat = new AudioFormat(8000.0f, 16, 1, true, false);
}catch(Exception e){
    System.out.println(e);
}
```

Koodiesimerkki 5. Ääniformaattien alustaminen

Kun MicReader-luokka on luotu, on sillä kaksi päätehtävää. Ensimmäinen on mikrofonin lukeminen suoratoistoa varten. Toinen on uusien äänitiedostojen nauhoittaminen, mutta tästä enemmän aliluvussa 3.3.6. Muilla MicReader-luokan metodeilla lähinnä ohjaillaan näiden kahden muun toimintaa.

Kun lähetetään kutsu MicReader-luokalle lukea mikrofonia suoralähetystä varten, annetaan metodille samalla näytteenottotaajuus. Näytteenottotaajuus kertoo, kuinka monta näytettä lähteestä otetaan sekunnissa. Näitä näytteitä käytettään vastaanottavassa päässä luomaan äänisignaali uudelleen. Esimerkiksi CD-levyt ja MP3-formaatti käyttävät näytteenottotaajuutena 44,1 kHz [20.]. Mikäli näytteenottotaajuutta ei anneta tai se on alle 32000 Hz, käytetään MicReader-luokan luonnissa annettua näytteenottotaajuutta. Näytteenottotaajuuden valitsemisen jälkeen siirrytään mikrofonin lukemisen alustukseen, josta koodiesimerkki 6.

```
try {
    DataLine.Info info = new DataLine.Info(TargetDataLine.class, broadcastFormat);
    mic = (TargetDataLine) AudioSystem.getTargetDataLine(broadcastFormat);
    mic = (TargetDataLine) AudioSystem.getLine(info);
    recording = true;
    mic.open(broadcastFormat);
    data = new byte[mic.getBufferSize()/5];
    mic.start();
} catch (Exception e) {
    System.out.println(e);
}
```

Koodiesimerkki 6. Mikrofonin lukemisen alustus

Ensimmäisenä alustamme linjan, jota mikrofoni käyttää. Tässä linjana toimii tietokoneen oletussisääntulo. Asetamme linjalle myös aikaisemmin määritellyn ääniformaatin. Seuraavaksi asetamme mikrofonintilaa ylläpitävän totuusarvon todeksi. Tätä arvoa käytetään myös muun muassa testaamaan, onko mikrofoni jo käytössä. Kun mikrofoni on varattu, avataan se ja määritetään puskurin koko datan lukemista varten. Viimeisenä aloitetaan mikrofonin lukeminen.

Kun mikrofonin lukeminen on aloitettu, siirrytään säikeeseen, joka lukee datan ulostulevaan tavutaulukkovirtaan. Tästä kertoo koodiesimerkki 7.

```
while (recording) {
    int byteCount = mic.read(data, 0, data.length);
    if (byteCount > 0) {
        out.write(data, 0, byteCount);
    }
}
```

Koodiesimerkki 7. Mikrofonin data ulostulevaan tavutaulukkovirtaan

Säikeessä luetaan mikrofonia, kunnes aikaisemmin todeksi määritelty totuusarvo muutetaan epätodeksi. Mikrofonin data luetaan taas aikaisemmin määriteltyyn puskuriin. Puskuri kirjoitetaan MicReader-luokan luonnin yhteydessä alustettuun ulostulevaan tavutaulukkovirtaan. Tätä tavutaulukkovirtaa Broadcast-olio käyttää tarvittaessa suoratoistossa.

Kun mikrofonin lukeminen halutaan lopettaa, kutsutaan sitä varten olevaa metodia. Metodi pysäyttää mikrofonin, sulkee sen ja asettaa mikrofonia lukevan toistorakennetta ohjaavan totuusarvon epätodeksi. Tästä kertoo koodiesimerkki 8.

```
public void stopMic(){
    mic.stop();
    mic.close();
    recording = false;
}
```

Koodiesimerkki 8. Mikrofonin lukemisen lopetus

3.3.6 Tiedostojen luonnin toteutus

Kun halutaan luoda äänitiedosto, tarvitsee aluksi luoda MicReader-luokka, jos sitä ei ole vielä luotu. Äänitiedostoa luodessa MicReader-luokka luodaan samalla tavalla, kun pelkän mikrofonin lukemisen yhteydessä. Tämä on esitetty aliluvussa 3.3.5.

MicReader-luokan alustamisen jälkeen voidaan kutsua nauhoituksen aloitusta. Nauhoituksen aloituksen kutsuminen vaatii nauhoituksessa käytettävän näytteenottotaajuuden ja tiedostonnimen. Mikäli näytteenottotaajuus on enemmän kuin 8000 Hz käytetään sitä tiedoston luonnissa. Seuraavaksi alustetaan mikrofoni. Mikrofonin alustus tapahtuu samalla tavalla, kuin mikrofonia luettaessa. Tämän voi katsoa aliluvussa 3.3.5 ja koodiesimerkistä 6.

Alustamisen jälkeen siirrytään säikeeseen toteuttamaan tiedoston luonti. Tästä kertoo koodiesimerkki 9.

```
try {
    ais = new AudioInputStream(mic);
    if (fileName != null){
        File newFile = new File(fileName + ".wav");
        AudioSystem.write(ais, fileType, newFile);
    } else {
        AudioSystem.write(ais, fileType, tempFile);
    }
} catch (Exception e) {
    System.out.println(e);
}
```

Koodiesimerkki 9. Äänitiedoston luonti

Ensimmäisenä alustetaan sisääntuleva äänivirta mikrofonista. Tämän jälkeen testataan, annettiinko tiedostolle nimi. Jos tiedostolle ei annettu nimeä, kirjoitetaan sisääntulevasta

äänivirrasta data MicReader-luokan sisäiseen väliaikaiseen tiedostoon. Jos tiedostolle annettiin nimi, luodaan tämänniminen ääniaaltotiedosto ohjelman juurikansioon. Sisääntulevan äänivirran data kirjoitetaan tähän tiedostoon.

Tiedoston luominen loppuu, kun mikrofoni suljetaan. Tämä onnistuu käyttämällä samaa metodia kuin mikrofonin lukemisen lopettamisessa.

3.4 Käyttöliittymän toteutus

Käyttöliittymä on toteutettu käyttäen javax.swing-kirjastoa. Käyttöliittymä on jaettu neljään luokkaan. Luokista on alla lyhyt kuvaus.

- **UserInterface** on pääluokka. Sen tehtävä on luoda UIController-luokan ilmentymä. UserInterface-luokka ajaa myös säiettä, jonka tehtävä on kertoa ohjaimelle, kun käyttöliittymä suljetaan x-painikkeesta.

- **UIController** on luokka, jonka tehtävä on alustaa View- ja Model-luokat sekä asettaa ne toisilleen. Tämän lisäksi se luontinsa yhteydessä kutsuu View-luokkaa luomaan graafisen käyttöliittymän kirjautumiselle.

- **View** on luokka, joka luo graafisenkäyttöliittymän. View-luokka luo käyttöliittymän kirjautumista sekä tämän onnistuessa ohjaimen käyttöä varten. Luokka myös päivittää ulkoasuaan Model-luokan kutsujen ja sisäisten kutsujensa perusteella. Käyttöliittymässä esitettyjen listojen käsittely ja nappien syöteittä tarkistava logiikka on myös sisällytetty View-luokkaan.

- **Model** on luokka, joka toimii käyttöliittymän ohjaimen-rajapintaa käyttävänä luokkana. Model-luokka sisältää metodeja, jotka toimivat kutsuina ohjaimelle ja palauttavat tarvittaessa ohjaimelta saatavia tietoja. Tämän lisäksi Model-luokka toteuttaa ohjaimen kaiuttimien tietoja sisältävää listaa lukevan säikeen, joka pitää käyttöliittymän kaiutinlistan ajan tasalla.

UserInterface- ja UIController-luokat ovat toteutukseltaan hyvin yksinkertaisia. UserInterface-luokka sisältää vain muutaman rivin koodia, jossa luodaan UIController, joka ei

vaadi mitään syötteitä. Tämä jälkeen luodaan säie, joka kutsuu UIController- ja Model-luokkien kautta metodia, jolla lopetetaan ohjaimen yhteys sekä suoratoistavat säikeet.

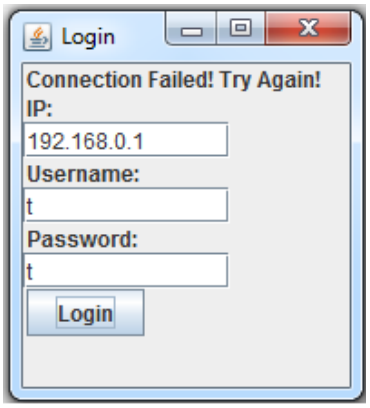
UIController-luokka luo ensimmäisenä View-luokan. Seuravaksi luodaan Model-luokka, jolle annetaan View-luokka. Tämän jälkeen View-luokalle annetaan Model-luokka. Viimeisenä kutsutaan View-luokan metodia, joka luo käyttöliittymän kirjautumiselle.

3.4.1 Kirjautumisen toteutus käyttöliittymässä

Kirjautuminen käyttöliittymässä alkaa UIController-luokan yhteydessä, kun sen luonnin lopussa kutsutaan View-luokkaa luomaan kirjautumisnäkymä. Kirjautumisnäkymä sisältää vaadittavista syötteistä kertovia tekstejä, tekstikentät IP-osoitteelle, käyttäjänimelle, salasanalla ja lopussa kirjautumispyynnön suorittavan napin, kuten voimme huomata kuvasta 3.

Kuva 3. Kirjautumisnäkymä

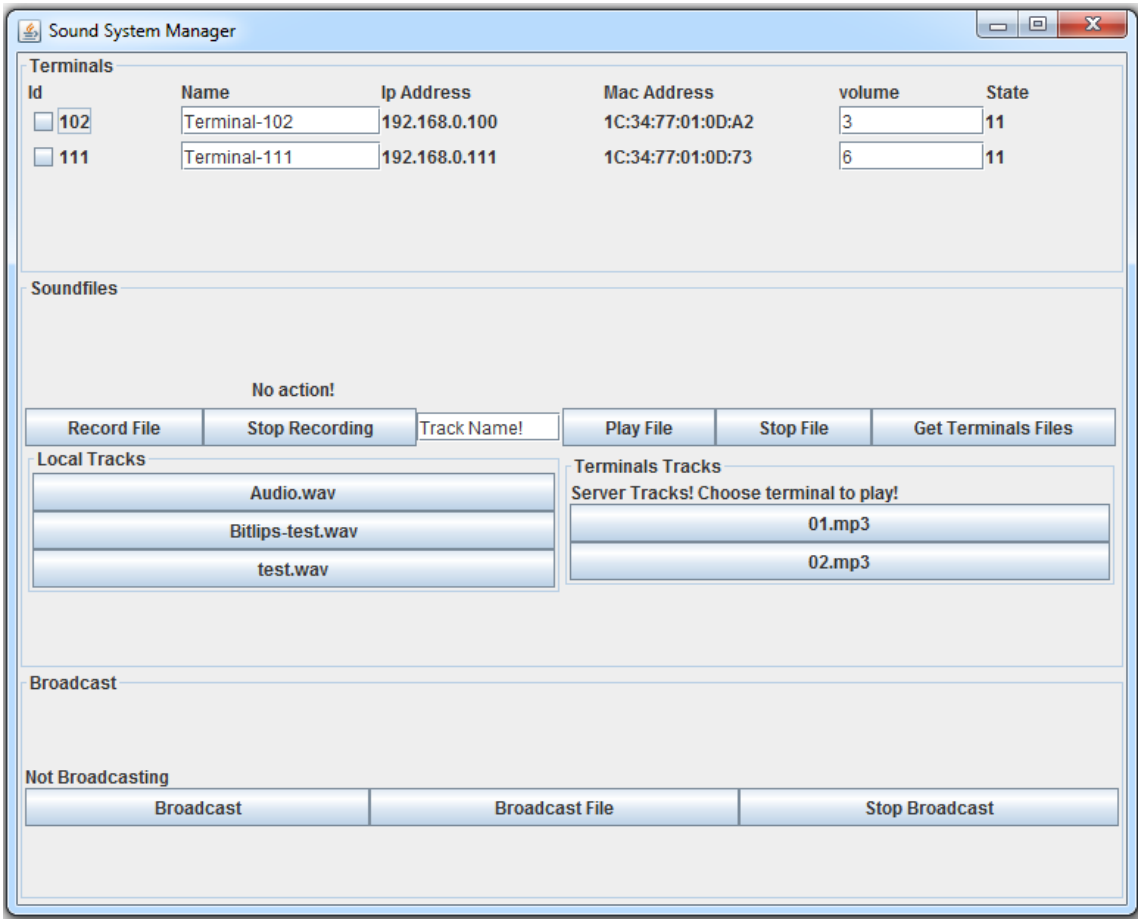
Kun kirjautumispyynnön lähettävästä napista painaa, lähettää View-luokka Model-luokan kautta pyynnön ohjaimelle luoda TCP-yhteyden IP-kenttään kirjoitettuun IP-osoitteeseen. Jos yhteyden luominen onnistui, lähetään ohjaimelle käyttäjänimi ja salasana. Vastauksena saadaan kokonaisluku, joka vastaa kirjautumisen onnistumista tai jotain kirjautumisessa ilmennyttä häiriötä. Häiriötapauksessa se ilmoitetaan kirjautumisnäkymän ylälaidassa tekstinä, kuten kuvassa 4.



Kuva 4. Epäonnistunut kirjautuminen

3.4.2 Ohjainpaneelin toteutus käyttöliittymässä

Jos kirjautuminen onnistuu, kutsuu se silloin View-luokkaa muuttamaan kirjautumisnäkymän ohjainpaneeliksi. Ohjainpaneeli sisällöstä saa parhaiten idean katsomalla kuvaa 5.



Kuva 5. Ohjainpaneeli

Ylimpänä löytyy lista kaiuttimista, joihin käyttäjällä on oikeus. Kaiuttimista kerrotaan niiden ID, nimi, IP-osoite, MAC-osoite, äänenvoimakkuus ja tila. Kaiuttimen pystyy valitsemaan aktiiviseksi ID-numeron viereisestä valintanappulasta. Ohjainpaneeli käyttää aktiivisena olevia kaiuttimia monessa toiminnossa. Nimen muuttaminen muuttaa ainoastaan kaiuttimen nimeä paikallisesti. Äänenvoimakkuuden muuttaminen lähettää Model-luokan kautta ohjaimelle äänenvoimakkuuden muuttamisen komennon. Tilan kohdalla oleva kokonaisluku kertoo palvelimelta saadun tilasta ilmoittavan kokonaislukuarvon.

Äänen voimakkuuden muuttaminen on toteutettu javax.swing.DocumentListener luokalla. Kun äänenvoimakkuus tekstikenttiä luodaan ohjainpaneelin, luonninyhteydessä asetetaan jokaiseen oma DocumentListener-olio, joka saa syötteenä oman kaiuttumensa ID-numeron ja monesko kaiutin se on listassa. Näiden tietojen perusteella se osaa lähettää oikeanlaisen kutsun ohjaimella. Tästä kertoo koodiesimerkki 10.

```
@Override
public void insertUpdate(DocumentEvent e) {
    int volumeInt = Integer.parseInt(volume[terminalNumber].getText());
    List<Terminal> tempTermi = new ArrayList<Terminal>();
    Terminal temp = null;
    for (Terminal termis : termi) {
        if (termis.getId() == id) {
            temp = termis;
        }
        tempTermi.add(temp);
    }
    if (temp != null) {
        model.changeVolume(volumeInt, tempTermi);
    } else {
        System.out.println("No available terminal");
    }
}
```

Koodiesimerkki 10. Äänenvoimakkuuden muutoksen kuuntelija

DocumetListener-olio kutsuu tätä metodia, kun tekstikenttään syötetään jotain. Tämän jälkeen otetaan syötetty luku ylös. Ohjain vaati antamaan kaiuttimet, joihin halutaan tehdä muutoksia kaiutinlistoina. Tämän takia luomme väliaikaisen kaiutin listan ja väliaikaisen kaiuttimen. Seuraavaksi etsimme kaikista kaiuttimista sen kaiuttimen, jolla on sama ID-numero, kun DocumentListener-oliolle annettiin. Kun oikea kaiutin on löytynyt, lisätään se väliaikaiseen listaan ja lähetetään annettu äänenvoimakkuus ja väliaikainen lista Model-luokan kautta ohjaimelle.

Ohjainpaneelin keskeltä löytyvät äänitiedostojen hallinnan painikkeet. Tämäkin on jaettu kahteen osaan. Vasemmalla ovat painikkeet tiedostojen äänittämiseen ja tekstikenttä nimeä varten. Näiden alta löytyvät kaikki tämänhetkiset paikalliset äänitiedostot. Oikealla ovat painikkeet kaiuttimien kappaleiden toistamiseen ja niiden hakemiseen sekä niiden alla palvelimen ja kaiuttimien kappaleet listana.

Tiedostojen äänittäminen ja muut painikkeilla ohjattavat toiminnot on toteutettu ActionListener-olioilla. Tiedostojen äänittäminen on näistä yksinkertaisimpia. Aluksi tarkistamme Model-luokan kautta ohjaimelta, että mikrofoni on vapaa. Seuraavana tarkistamme, että kappaleelle on annettu nimi. Tämän jälkeen voimme luoda tämän nimisen tiedoston ohjaimen juurihakemistoon. Viimeisenä kerromme Model-luokan kautta ohjaimelle aloitta äänittämisen. Kun haluamme lopettaa tiedoston äänittämisen, tarkis-

tamme, että äänittäminen on käynnissä. Jos äänittäminen on käynnissä, kerromme ohjaimelle lopettaa sen. Tämän jälkeen päivittämme paikallisten kappaleiden listan. Se on toteutettu tyhjentämällä lista ja hakemalla kaikki uudelleen.

Kaiuttimien kappaleiden toistaminen alkaa hakemalle aluksi haluttujen kaiuttimien kappaleet. Kun haetaan kaiuttimien kappaleita, käytetään kaiuttimien valinnassa kaiutin valikon valintapainikkeita. Koodiesimerkki 11 esittää, kuinka painikkeet hakevat valitut kaiuttimet listaksi komentoja varten.

```
int idsCount = 0;
for (int i = 0; i < termi.size(); i++) {
    if (ids[i].isSelected()) {
        idsCount++;
    }
}
if (idsCount != 0) {
    int terminalIds[] = new int[idsCount];
    int idCount = 0;
    for (int i = 0; i < termi.size(); i++) {
        if (ids[i].isSelected()) {
            terminalIds[idCount] = termi.get(i).getId();
            idCount++;
        }
    }
    List<Terminal> tempTermi = new ArrayList<Terminal>();
    Terminal temp = null;
    for (Terminal termis : termi) {
        for (Integer i : terminalIds) {
            if (termis.getId() == i) {
                temp = termis;
                tempTermi.add(temp);
            }
        }
    }
    model.getTerminalTracks(tempTermi);
```

Koodiesimerkki 11. Kaiutinvalinta listojen luonti

Aluksi tarkistamme, kuinka moni kaiuttimista on valittu. Jos edes yksi kaiutin on valittu, siirrytään varsinaisen koodin suoritukseen. Ensimmäisenä luomme kokonaismuuttujataulukon, jossa on yhtä monta muuttujaa kuin valittuja kaiuttimia. Tämän jälkeen taulukko täytetään valittujen kaiuttimien ID-numeroilla. Seuraavana luomme väliaikaisen kaiutinlistan, jonka täytämme valittuja kaiuttimia vastaavilla kaiutin-oliolla. Kun kaiutinlista vali-

tuista kaiuttimista on luotu, voidaan sitä käyttää ohjaimelle lähetettävässä kutsussa. Koodiesimerkissä 11 käytetään kaiutinlistaa hakemaan näiden kaiuttimien paikalliset kappaleet.

Kun valittujen kaiuttimien kappaleet on haettu, tallentuvat ne kaiuttimin-olioiden kappalelistoihin. Käyttöliittymä poistaa sillä hetkellä esitetyn kappalelistan ja asettaa siihen viimeisimpänä haetun kappalelistan. Tätä kappalelistaa käytetään, kun halutaan toistaa kaiuttimeen tallennettuja kappaleita. Valittujen kaiuttimien valinta tapahtuu samalla tavalla, kun edellä on esitetty, ja kappaleen valinta tapahtuu koodiesimerkin 12 mukaisella tavalla.

```
if (terminalTracksButtons.getSelection() != null) {
    model.playTerminalTrack(Integer.parseInt(terminalTracksButtons.getSelection().getActionCommand()), tempTermi);
}
```

Koodiesimerkki 12. Toistettavan kappaleen valinta.

Ensimmäisenä tarkistetaan, että joku kappaleista on valittu. Tämän jälkeen lähetetään Model-luokan kautta ohjaimelle komento, joka sisältää muuttujina valitun kappaleen ja valitut kaiuttimet. Valittu kappale ilmoitetaan sen järjestysnumeron kokonaisnumerona, joka on tallennettu ActionCommand-muuttujaan.

Kaiuttimiin tallennetun kappaleen toisto loppuu, kun kappale loppuu annettaessa lopetuskomento. Lopetuskomento lähettää ohjaimelle valitut kaiuttimet ja lopettaa toiston näissä.

Ohjainpaneelin alhaalta löytyvät suoratoistoa ohjaavat painikkeet. Tavallinen suoratoisto toimii hyvin samalla tavalla kuin koodiesimerkki 11. Erona on vain kutsu, jossa kaiutinlistat annetaan Model-luokan suoratoiston aloittavalle metodille. Sama pätee myös suoratoiston lopettamisessa, jossa kutsuna toimii suoratoiston lopetus. Kun suoratoisto toteutetaan tiedostosta, on valittujen kaiuttimien toteutus taas sama. Tiedoston valinta toimii hyvin samalla tavalla kuin koodiesimerkissä 12, mutta tällä kertaa ActionCommand-muuttujaan on tallennettu tiedostonnimi.

Ohjainpaneelin luonnin yhteydessä luodaan palvelimelta tulevia muutoksia päivittävä säie. Säikeessä toimii toistorakenne, joka jatkuu, kunnes kutsutaan ohjelman lopetusta,

kuten painamalla oikean yläkulman x-painiketta. Toistorakenteessa säie tarkistaa, että kaiutinlista on olemassa, kuten huomaamme koodiesimerkistä 13.

```
while (online) {
    if (termi != null) {
        List<Terminal> tempTermi = controller.getTerminals();
        view.update(tempTermi);
        try {
            Thread.sleep(1000);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Koodiesimerkki 13. Kaiutinlistan päivittäminen

Jos kaiutinlista löytyi, pyydetään ohjaimelta väliaikainen kaiutinlista, jossa olisivat mahdolliset palvelimelta tulleet muutokset. Väliaikainen palvelinlista annetaan View-luokan update-metodille, joka päivittä käyttöliittymän esittämää kaiutinlistaa. Lopuksi säikeessä odotetaan sekunti ennen seuraavaa toistoa.

4 Järjestelmän toiminnan arviointi

Järjestelmä toimii kohtuullisen hyvin. Rajapinnan kutsut pystyttiin suorittamaan ilman kaatumisia. Järjestelmässä ei myöskään ilmennyt isoja viiveitä. Isoimmat ongelmat liittyivät ohjaimen suoratoistoon toteutukseen, jonka laadussa oli hiukan ongelmia. Tästä kerrotaan enemmän aliluvussa 4.1.

4.1 Ohjaimen toiminnan arviointi

Ohjaimen toiminnassa onnistuttiin saavuttamaan kohtuullinen taso. Kaikki aliluvussa 3.1 listatut toiminnot pystyttiin toteuttamaan ohjaimella ilman ohjelman kaatumista. Isoimpia ongelmia, joita en saanut ohjelmassa ratkaistua, olivat suoratoistonlaatuun liittyvät ongelmat. Näistä isoin oli välillä tapahtuva tiedostosta toistettaessa äänen nykiminen. Toinen iso ongelma oli lähetysviiveen kasvaminen, kun suoratoiston mikrofoni jätettiin auki.

Kuinka paljon tämä varsinaisesti haittaa käytössä, ei ole tiedossa, koska jatkuvassa puheessa ei viivettä huomannut. Viimeisenä huomasin välillä suoratoistojen tukkivan kaiuttimia niin, että ne tarvitsivat uudelleen käynnistämistä. Tämä saattaa tosin johtua myös laitteista, koska en saanut ikinä toistettua ongelmaa samalla tavalla, mutta ongelma ilmeni kerran testatessani datansiirtoa OBT-ohjelmilla.

Parannusmahdollisuuksia ohjaimessa olisi voinut olla suoratoiston lähetystavassa ja -muodossa. Etenkin tiedostojen suoratoistossa ainoat testatut formaatit olivat WAV ja MP3. Esimerkiksi FFmpeg-pakkauksenhallinnan avulla olisin voinut kokeilla useampia formaatteja. Olisin voinut myös kokeilla toteuttaa suoratoistoa käyttäen RTP-protokollaa. Totesin molemmat edellä mainituista parannusehdotuksista kuitenkin liian työläiksi, koska ne eivät taannet parannusta suoratoistossa. Esimerkiksi kaiuttimien toistamista ääniformaateista ei ollut mitään dokumentaatiota, jota olisi voinut käyttää sen valinnassa.

4.2 Käyttöliittymän toiminnan arviointi

Testikäyttöön tarkoitetuksi käyttöliittymäksi käyttöliittymä toimii hyvin. En huomannut viimeisissä testeissä ongelmia tai kaatumisia. Mahdollisia ongelmia käyttöliittymällä voi tulla isoissa järjestelmissä, sillä suunnittelin käyttöliittymän toimimaan vain pienissä testiympäristöissä. Suurien järjestelmien ongelmia voi olla esimerkiksi, ettei käyttöliittymä pysty esittämään kaikkia kaiuttimia kerralla. Samanlaiseen ongelmaan voidaan törmätä, jos palvelimelle, kaiuttimiin tai paikallisesti tallennetaan liikaa kappaleita.

Parannusmahdollisuuksia käyttöliittymässä voisi olla lisätä listoihin selaamisen mahdollista liukupalkki. Myös kaiuttimien tilasta kertovan tilanumeron olisi voinut korvata tilaa kuvaavalla tekstillä. Kappaleiden nimensyöttökentän olisi voinut sijoittaa ponnahdusikkunaan ahtaan tekstikentän sijaan. Nämä olivat kuitenkin mielestä omaan testiympäristööni epäoleellisia ja mahdollisesti aikaa vieviä lisäyksiä. Käyttöliittymän olisi voinut myös toteuttaa mahdollisesti toimimaan selaimessa, jolloin sille olisi voinut olla myöhemmin muutakin tarvetta kuin rajapinnan testaaminen.

5 Johtopäätökset

Työssä päästiin sen kumpaankin johdannossa esiteltyyn tavoitteeseen. Onnistuttiin toteuttamaan toimiva rajapinta OBT-kuulutusjärjestelmään ja tätä rajapintaa käyttävä työpöytäsovellus. Työssä esiteltiin myös yleisesti, missä suoratoistoa käytetään ja kuinka sen voi toteuttaa Javassa. Esittelin myös, mitä mahdollisia ongelmia ja ratkaisuja tähän liittyy.

Työn toteutuksen pohjalta saatiin seuraavat johtopäätökset. Javalla pystyy toteuttamaan äänen reaaliaikaiseen suoratoistoon kykenevän ohjelman. Tähän löytyvät kaikki tarvittavat kirjastot Javan versiosta 8, mutta parempaan lopputulokseen päästään todennäköisesti, kun otetaan käyttöön kolmannen osapuolen kirjastoja. Tämä on jo pelkästään välttämätöntä, jos halutaan käyttää useampia ääniformaatteja.

Kun toteutetaan rajapintaa jo valmiiseen järjestelmään, järjestelmän hyvä tuntemus on tärkeää toteutuksen kannalta. Järjestelmän hyvä tuntemus muun muassa vähentää aikaa, joka vaaditaan rajapinnan komentojen selvittämiseen. Hyvä järjestelmän tuntemus mahdollistaa myös toteuttamaan rajapinnalla ominaisuuksia, joihin sitä ei ole alun perin tarkoitettu, kuten tässä tapauksessa tiedostojen suoratoisto. Parempi järjestelmän tuntemus olisi tässä tapauksessa myös mahdollistanut paremman äänenlaadun.

Yleisesti suoratoiston kannalta tultiin johtopäätökseen, että oikean verkkoprotokollan valinta tietyntyyppiseen suoratoistoon on hyvin tärkeää. Eli käytetään UDP-protokollaa tai jotain sitä käyttävää protokollaa, kuten RTP reaaliaikaisessa suoratoistossa. TCP-protokollaa taas käytetään tiedostojen suoratoistossa. Tärkeää on myös oikeanlaisen pakkauksenhallinnan ja formaatin käyttäminen tietylle datalla. Tärkeintä on kuitenkin, että lähettävä ja vastaanottava pää käyttävät samaa pakkauksenhallintaa.

Työ on ollut Avackilla jatkokehityksessä, jossa on saatu ratkaistua äänen suoratoiston laatuun ja viiveeseen liittyvät ongelmat. Samalla ohjaimelle on saatu luotua selainpohjainen ratkaisu, josta puhuin aliluvussa 4.2.

Lähteet

1. Skype protocol. Nettiartikkeli. Margaret Rouse. <http://searchunifiedcommunications.techtarget.com/definition/Skype-protocol>. Luettu 11.1.2017

2. Spotify – Large Scale, Low Latency, P2P Music-on-Demand Streaming. Gunnar Kreitz, Fredrik Niemelä. Tutkimusraportti. Julkaistu osana Proceedings of IEEE P2P 2010

3. Netflix planning implement p2p streaming. Nettiartikkeli. Ashley Allen. <http://www.eteknix.com/netflix-planning-implement-p2p-streaming/>. Luettu 13.12.2016

4. Samarat Ganguly, Sudeept Bhatnaget. VoIP: Wireless, P2P and New Enterprise Voice over IP. John Wiley and Sons Ltd., Hoboken, Yhdysvallat, 2008.

5. Global IP Sound. Nettiartikkeli. <http://www.voip-info.org/wiki/view/Global+IP+Sound>. Luettu 13.1.2017

6. What You Might Not Know About Bluetooth, Reasons Why Bluetooth Can Reduce Audio Quality. Brent Butterworth. Nettiartikkeli. <https://www.lifewire.com/what-to-know-about-bluetooth-3134591>. Luettu 13.1.2017

7. LGPL. Nettiartikkeli. <http://www.webopedia.com/TERM/L/LGPL.html>. Luettu 14.1.2017

8. About FFmpeg. Nettiartikkeli. <https://ffmpeg.org/about.html>. Luettu 14.1.2017

9. efflux. Ohjelmiston kuvaus. <https://github.com/biasedbit/efflux>. Luettu 16.1.2017

10. Benchmark: How Misusing Streams Can Make Your Code 5 Times Slower. Alex Zhitnitsky. Nettiartikkeli. <http://blog.takipi.com/benchmark-how-java-8-

lambdas-and-streams-can-make-your-code-5-times-slower/>. Luettu 14.1.2017

11. What are AIFF, AIF, & AIFC Files?. Tim Fisher. Nettiartikkeli. <https://www.lifewire.com/aiff-aif-aifc-files-2619569 >. Luettu 15.1.2017

12. SND Format. Nettiartikkeli. <http://www.abyssmedia.com/formats/snd-format.shtml>. Luettu 15.1.2017

13. Ogg Vorbis. Margaret Rouse. Nettiartikkeli. <http://whatis.techtarget.com/definition/Ogg-Vorbis>. Luettu 15.1.2017

14. What bitrate does Spotify use fot Streaming?. Nettiartikkeli. <https://support.spotify.com/us/using_spotify/search_play/what-bitrate-does-spotify-use-for-streaming/>. Luettu 15.1.2017

15. Introduction to JavaFX Media. Cindy Castillo. Nettiartikkeli. <http://docs.oracle.com/javafx/2/media/overview.htm>. Luettu 16.1.2017

16. Real-Time, Low Latency Audio Processing In Java. Nicolas Juillerat, Stefan Müller Arisona, Simon Schibiger-Banz. Tutkimusraportti.

17. ALSA project – the C libary recerence. Nettiartikkeli. <http://www.alsa-project.org/alsa-doc/alsa-lib/>. Luettu 16.1.2017

18. easyUML -plugin detail. Liitännäisen kuvaus.< http://plugins.netbeans.org/plugin/55435/easyuml>. Luettu 19.1.2017

19. Primitive Data Types. Nettiartikkeli. <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>. Luettu 23.1.2017

20. Digital Audio Basics: Sample Rate and Bit Depth. Nettiartikkeli. <http://www.presonus.com/news/articles/sample-rate-and-bit-depth>. Luettu 24.3.2017